TLN - Mazzei

Mario Bifulco

a.a. 2022-2023

Indice

1 info utili

Caratteristiche del linguaggio naturale:

- Discretezza
- Ricorsività
- Dipendenza
- Località

Inoltre il linguaggio non è standard e si evolve con neologismo

L'uomo cerca l'uomo anche quando non c'è, problema pareidolia, quindi molti test di valutazione del linguaggio naturale sono intrinsecamente sbagliati

Per gestire il linguaggio naturale si assume di poter lavorare a livelli (fonetica, morfologia, sintassi, semantica, pragmatica, discorso)

Nella morfologia abbiamo classi chiuse e aperte

Nel NER-tagging si usa BIO-tagging con i tag PER, LOC, ORG, GPE

Nella pragmatica avviene il task di **anafora resoltion**, in cui si fa mapping tra riferimenti "generici" e oggetti menzionati in precedenza

Il dialogo è un'attività collaborativa motivata da goal

2 Parsing grammatiche a dipendenze

Parsing a dipendenze == relazioni tra coppie di parole (relazione asimmetrica). Composta da una **testa/dominante** e la **dipendenza/dominato**. Si creano alberi a sviluppo orizzontale. Con questo formalismo non ho più simboli non terminali

Per riuscire a gestire alcune dipendenze complesse *Tesnière* fa due ipotesi

1. Alcune preposizioni (come *di*) sono funzioni che trasformano un nome in aggettivo
2. Le congiunzioni rendono le relazioni simmetriche

Le grammatiche a dipendenze sono più generiche (frasi uguali ma con ordine diverso delle parole restituiscono lo stesso albero), simili al funzionamento umano, semplici nella rappresentazione e sono più indipendendi dalla lingua visto che catturano le relazioni. Si possono usare diversi algoritmi (varianti di CKY o CFG probabilistiche, programmazione dinamica), inoltre sono disponibili diverse tecniche per il parsing:

- MST
- conversione con teoria $\overline{X}$
- soddisfacimento di vincoli (elimino a cascata le dipendenze che non soddisfano certi vincoli)
- strategie greedy guidate da machine learning

Per la valutazione dei parser a dipendenze si usa la **labelled attachment score** (percentuale di sottoalberi creati correttamente). I parser per le grammatiche a dipendenze visti a lezione sono:

MALT Parser deterministico (scelte greedy), algoritmo bottom-up con ricerca depth-first (non serve backtracking perché usa le probabilità), usa un oracolo probabilistico. In questo contesto *deterministico* significa che per ogni frase restituirà sempre un albero (a differenza di CKY). Per lavorare utilizza una descrizione a stati che comprende 1. stack di lavoro, 2. input buffer, 3. parser con oracolo per associare l'azione (leftarc, rightarc o shift). Questa struttura è nota come parsing basato su transizioni

- **Shift**: prende la parola in input e la mette sullo stack rimuovendola dal buffer
- **Left**: crea la dipendenza (a, b) tra la parola in input (a) e la coma dello stack (b), poi fa pop dello stack
- **Right**: crea la dipendenza (b, a), rimuove a dalla lista, poi fa pop dallo stack mettendo b a inizio lista

Solitamente al posto di 3 archi l'oracolo ne utilizza 2n+1, in modo da associare all'arco l'etichetta. Per addestrare l'oracolo si possono usare regole scritte a mano o addestrare un modello tramite i TreeBank.

Si generano un numero di stati pari al cubo delle parole in input

TUP Approccio tramite soddisfacimento di vincoli, algoritmo bottom-up e depth-first, oracolo basato a regole. Usava regole di chunking nominale, coordinazione (per rimuovere l'ambiguità della congiunzione) e verb-subcat (tassonomia per le regole verbali, usata per collegare in chunk). Le dipendenze potevano essere morfo-sintattiche, sintattico-funzionali o semantiche

2.1 CKY probabilistico

Usa PCFG (probabilistic context free grammar), ricerca bottom-up, left-to-right con programmazione dinamica e beam search, usa un oracolo probabilistico. La probabilità dell'albero è il prodotto delle probabilità delle produzioni (calcolate dai TreeBank, che definiscono induttivamente anche le grammatiche).

$$P(\alpha \to \beta|\alpha) = \frac{Count(\alpha \to \beta)}{Count(\alpha)}$$

Questo approccio inibisce la correttezza a favore della velocità dell'algoritmo. Nella tabella tridimensionale la singola cella memorizza quali sono state le produzioni che hanno prodotto il sottoalbero con una certa probabilità, Memorizzando così solo i sottoalberi con probabilità più alta, non si possono più ottenere più alberi validi

Usare le probabilità aumenta la velocità ma crea problemi quando si hanno frasi molto simili grammaticalmente ma fortemente diverse come significato, per ovviare a questo problema si può lessicalizzare la grammatica (shift verso la semantica), così ho molte più informazioni, ma anche molte più regole.

Si generano un numero di stati proporzionale al cubo della frase in input moltiplicato per il cubo del numero di produzioni

2.2 Algoritmo su grafo MST

Costruisco gli MST (alberi completamente connessi) e poi valuto le frequenze di occorrenza delle parole, ottenendo delle probabilità sugli archi. Viene scelto MST più probabile, poi un classificatore si occupa di tipare gli archi. In generale si generano stati propolzionalmente al quadrato del numero di parole in input

3 Lambda-calcolo

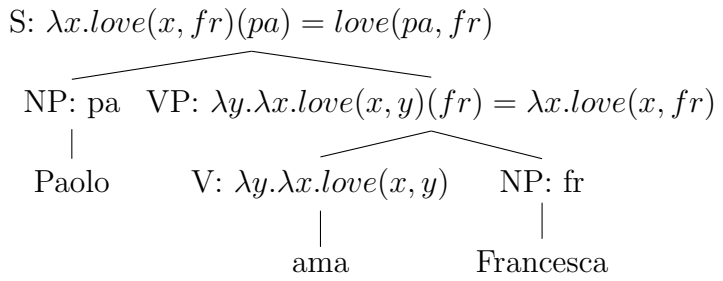
Nel contesto della semantica, per gestire casi con frasi incomplete si espande la logica del primo ordine FoL con il $\lambda - calcolo$. Questo formalismo è coerente con l'ipotesi di Montague, per la quale il linguaggio naturale è uguale a trattare un linguaggio formale (semantica formale). La logica $\lambda - FoL$ funziona per il principio composizionale di Fredge (semantica composizionale)

L'idea è che la semantica composizionale + la semantica lessicale permettono ragionamento anche su parole sconosciute

Il significato di una frase è dato dal significato delle parti e di come queste sono combinate

Si usa l'operatore λ per il binding delle variabili libere, formalismo delle **augmentend CFG**, grammatiche ad attributi. Per eseguire la beta-reduction: 1. elimino λ, 2. rimuovo l'argomento, 3. sostituisco la variabile libera con l'argomento

Oltre alla frasi incomplete bisogna cambiare la grammatica per rendere gestibili e non ambigui i quantificatori. Tramite il meccanismo di *type raising* si possono gestire anche gli articoli dei nomi propri e i verbi transitivi. Inoltre reificando le funzioni rendo possibile avere una grammatica robusta rispetto al numero di avverbi (stile neo-Davidsoniano) Queste astrazioni permettono non solo di costruire le funzioni, ma anche di fare inferenza sulla base di conoscenza.



Il problema con questo albero è che non so se fare andare l'argomento a destra o sinistra. Si può scrivere in modo esplicito nelle produzioni della grammatica, ma in questo modo ci limitiamo a gestire un solo tempo verbale. Per una maggiore flessibilità si usa una struttura costituita da **Utterance** (quando viene detta la frase), **Reference time** e **Event time**. In questo modo possiamo gestire tutti i casi soggetto, verbo e complemento oggetto.

3.1 Articoli e sostantivi con lambda calcolo

Gli articoli sono più complessi da gestire, tuttavia si può fare *reverse engineering* dalle frasi per ricavare le regole

$$\lambda P.\lambda Q.\exists z(P(z) \wedge Q(z)) \rightarrow un$$

Per permettere a questa regola di funzionare serve anche *type raising* (scambia la funzionalità di chi sta a sinistra e chi sta a destra), ottenenedo una grammatica in cui i verbi hanno una propria definizione, i nomi propri un'altra e gli articoli una terza:

$uomo ::= \lambda x.man(x)$, $Paolo ::= \lambda P.P(paolo)$,
$corre ::= \lambda x.run(x)$, $ama ::= \lambda R.\lambda x.R(\lambda y.love(x, y))$

L'importante quando si modifica la grammatica è mantenere le proprietà di composizionalità e sistematicità (la grammatica deve funzionare in generale, non su singoli esempi)

4 CKY

L'algoritmo CKY (Cocke Kasami Younger) permette di calcolare se una frase è generabile con una data grammatica, inoltre restituisce tutti i possibili alberi sintattici di quella frase (con la struttura a costituenti). La complessità asintotica è di $O(n^3)$, dove n è il numero di parole che compongono la frase di input. Per funzionare l'algoritmo lavora su grammatiche context free trasformate in forma normale di Chomsky, CNF. Usa un parsing bottom-up, left-to-right e gestisce la memoria tramite programmazione dimanica, l'oracolo è a regole (usa le produzioni della grammatica).

Una cella della matrice contiene tutte le radici dei sottoalberi generabili da quel punto. Una grammatica CNF significa che non deve avere produzioni che da un non terminali vanno allo spazio vuoto e non deve avere produzioni non binarie.

Algorithm 1 CKY-parse

Require: words
Require: G
  for j $From$ 1 To $LENGTH(words)$ **do**
    for $A|A \rightarrow words[j] \in G$ **do**
      $table[j-1, j] \leftarrow table[j-1, j] \cup A$
    for i $From$ $j-2$ To 0 **do**
      for k $From$ $i+1$ To $j-1$ **do**
        for $A|A \rightarrow BC \in G \wedge B \in table[i,k] \wedge C \in table[k,j]$ **do**
          $table[i, j] \leftarrow table[i, j] \cup A$
  return table

j effettua il loop delle colonne, i riempie la riga i corrispondente alla colonna j e k effettua il loop delle possibili produzioni della grammatica. Per via della sua complessità asintotica questo tipo di parsing è troppo lento per usi real time, per cui spesso si opta per strategie di pruning probabilistiche, risultati parsiali

5 Ambiguità sintattica e semantica

Il linguaggio naturale, a differenza dei linguaggi formali, è caratterizzato dall'ambiguità, essa permette una comunicazione molto più efficiente, ma aumenta la difficoltà di gestione per i sistemi automatici. Esistono due tipi principali di ambiguità, l'ambiguità sintattica e l'ambiguità semantica.

L'ambiguità sintattica è legata alla struttura delle frasi, in questa classe troviamo l'*attachment ambiguity*, ovvero l'ambiguità legata a come attaccare i componenti nell'albero sintattico (**mangio la pizza con le acciughe**), e l'ambiguità di *coordinazione*, ovvero legata alle congiunzioni (**si può mangiare e bere qualcosa**). Un caso tipico dell'*attachment ambiguity*, è la *PP-attachment*, in cui ho una frase preposizionale dopo il verbo (**ho visto una ragazza con il telescopio**)

Nell'ambiguità sintattita l'ambiguità è nella costruzione dell'albero, ma una volta disambiguato la struttura risultante è monoreferenziata a livello semantico

L'ambiguità semantica nasce invece quando la frase ha un solo albero costruibile, ma più formule FoL associabili (**tutti gli uomini amano una donna**), in questo esempio in base all'ordine con cui dispongo i quantificatori cambio drasticamente il significato di quanto letto.

6 Espressività delle lingue naturali

Negli anni 50 Chomsky vuole teorizzare una struttura formale per le lingue naturali (competence), nascono così le tecniche di parsing a costituenti (simboli non terminali della grammatica). Per l'ipotesi di Loebell un costituente agisce nella frase come un'unità di significato (è un gruppo si parole contigue nella frase). Negli stessi anni teorizza anche una gerarchia di grammatiche (al crescere del potere espressivo cresce anche la complessità computazionale).

- Grammatiche lineari: $A \to aB$ - $(ab)^n$
- Grammatiche context-free: $S \to aSb$ - $a^n b^n$
- Grammatiche mildly context sensitive: $CB \to f(C, B)$ - $a^n b^n c^n$
- Grammatiche context sensitive: $Caa \to aaCa$ - a^{2^n}
- Grammatiche tipo zero: $\psi \to \theta$ - L_{diag}

Chomsky inizialmente usava le grammatiche libere dal contesto per formalizzare la lingua inglese, studi successivi hanno dimostrato che alcune lingue, come il tedesco svizzero, hanno dipendenze incrociate (cross-serial) e quindi non possono essere espresse tramite grammatiche CF.

Per superare i limiti delle grammatiche CF, ma al comtempo gestire la complessità computazionale, vengono teorizzate una serie di grammatiche alternative

Grammatiche Mildly context sensitive Teorizzate da Joshi nel 1985, sono un'estensione delle CF che permettono sia dipendenze nested che incrociate. Queste grammatiche sono parsificabili in tempo polinomiale, inoltre mantengono la proprietà di crescita costante (estendendo una frase corretta ottengo sempre una frase valida)

Tree adjoining grammars Teorizzate sempre da Joshi nel 1975, lavorano sugli innesti degli alberi sintattici. Possono essere lessicalizzate, ovvero fattorizzano la ricorsione con operazioni di adjoining sull'albero sintattico, questo estende il dominio della località

La struttura fondamentale è l'albero e non la lista, ci sono alcune operazioni fondamentali

- sostituzione (cambio una foglia con un albero)
- adjoining (innesto un albero dentro un altro albero, in questo modo aumento il potere generativo)

Combinatory categorial grammars Steedman 1985 e Satta 2010, approccio bottom-up (solitamente le grammatiche generative sono top-down). Il principio è partire dalle foglie (categorie), si associa ad ogni parola una categoria e si cercano altre categorie per combinarsi

Es. amare è un elemento che cerca qualcosa a destra (soggetto) e qualcosa a sinistra (oggetto)

Head grammars 1984, Pollard

Linear indexed grammars 1985, Gazdar

7 HMM vs MEMM

Hidden Morkov model e Maximum entropy Markov model sono due sistemi stocastici utilizzati per il PoS tagging

Usano entrambi l'algoritmo di Viterbi

Algorithm 2 Algoritmo di Viterbi

Require: observation on len T
Require: state-graph on len N
create a path probability matric viterbi[N+2, T]
for each state s from 1 to N **do**
$viterbi[s, 1] \leftarrow a_{0,s} * b_s(o_1)$
$backpointers[s, 1] \leftarrow 0$
for each time step t from 2 to T **do**
for each state s from 1 to N **do**
$viterbi[s, t] \leftarrow max_{s'=1}^{N} viterbi[s', t-1] * a_{s',s} * b_s(o_t)$
$backpointer[s, t] \leftarrow argmax_{s'=1}^{N} viterbi[s', t-1] * a_{s',s}$
$viterbi[q_F, T] \leftarrow max_{s=1}^{N} viterbi[s, T] * a_{s,q_F}$
$backpointer[q_F, T] \leftarrow argmax_{s=1}^{N} viterbi[s, T] * a_{s,q_F}$
return best-path

HMM Algoritmo generativo in cui si cerca di massimizzare $\hat{t}_1^n = argmax_{t_1^n} P(t_1^n|w_1^n)$. Per la fase di modelling si usano le regole Bayesiane per semplificare l'equazione. Oltre alla regola di Bayes per il calcolo delle probabilità condizionate si applica l'ipotesi Markoviana per ridurre la storia considerata (rende la computazione più efficiente)

$$\hat{t}_1^n = argmax_{t_1^n} P(w_1^n|t_1^n)P(t_1^n) \approx argmax_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

In questo modo il sistema apprende 2 probabilità:

- Stimare il tag data la parola
- Stimare il tag dato il tag precedente (storia)

Il learing viene fatto partendo da un corpus annotato manualmente

Per il decoding si usa l'algoritmo di Viterbi per creare una matrice con la programmazione dinamica che memorizza solo i percorsi più promettenti, in questo modo domiamo la complessità esponenziale che comporterebbe l'analisi di tutti i percorsi possibili, $v_t(j) = max_{i=1}^{N} v_{t-1}(i)a_{ij}b_j(o_t)$, dove $v_{t-1}(i)$

le probabilità dei path di Viterbi al tempo precedente, a_{ij} è la probabilità di transizione da q_i a q_j, $b_j(o_t)$ è la probabilità di osserare il simbolo o_t dato lo stato corrente j

Problemi HMM Le HMM si possono estendere per considerare storie più lunghe, ma si rischiano problemi di sparsity nelle rappresentazioni e quindi di avere informazioni poco rilevanti per stiamare le probabilità. Per gestire questi casi si possono combinare le probabilità degli n-grammi (peso storie da dimensione n a dimensione 0 con un fattore di probabilità λ)

Inoltre questo approccio è case sensitive, quindi aumenta di molto il numero di parole nel dizionario (almeno nella sua versione base)

MEMM Modello discriminativo alternativo alle HMM, cambio prospettico, le parole condizionano lo stato (con le HMM era lo stato a condizionare l'apparizione di una parola)

Per il modelling si identificano feature eterogenee (grande vantaggio rispetto alle HMM che avevano solo la storia) per combinarle in modo probabilistico, con l'obiettivo di avere una distribuzione quanto più uniforme possibile

$$P(y|x)? = \sum_{i=1}^{N} w_i f_i$$

$$? = w \cdot f$$

$$p(y = c|x) = p(c|x) = \frac{1}{Z} exp(\sum_i w_i f_i)$$

Il learning viene effettuato tramite multinomial logistic regression, e tramite la formula $\hat{w} = argmax_w \sum_j logP(y^{(j)}|X^j)$ si riduce ad un problema di ottimizzazione convessa

Per il decoding si usa anche in questo caso viterbi

Problemi MEMM La sparseness si presenta anche qui, inoltre ho il problema di gestire le parole sconosciute (posso usare smoothing, probabilità uniforme dei tag, guardare dizionari esterni o usare euristiche)

Confronto Sul modelling le MEMM permettono maggiore espressività, sul learning però le HMM risultano molto semplici da addestrare (basta contare e calcolare le probabilità), nel decoding entrambi i metodi usano Viterbi

8 Architettura NLG

Le 3 sezioni sevono per gestire le due fonti di conoscenza (che ricoprono ruoli molto diversi), ovvero la conoscenza sul dominio (universale) e la conoscenza del linguaggio (specifica per la frase che voglio generare)

Text planning Content determination e Discourse planning, faccio ampio uso della conoscenza di dominio per imabastire la frase da generare. Si usano formule logiche e l'output è un text plan

- Content determination: si decide cosa dire (Es. treno, destinazione e orario di partenza)
- Discourse planning: si creano le realazioni tra frasi (conceptual grouping -Es. Il prossimo treno è il treno A e parte alla 10- o rhetorical relationships -Es. Il treno A parte alle 10 ed è uno dei 20 treni per Glasgow-)

Sentence planning Effettuo sentence aggregation, lexicalizazion e referring expression generation, usando sia la conoscenza di dominio che la conoscenza della lingua per cui devo generare la frase. Si usano gli alberi e si genera il sentence plan/plans

- Sentence aggregation: si uniscono le frasi per avere un discorso coeso e fluido
- Lexicalisation: Si decidono le parole da usare per esprimere concetti e relazioni del dominio, inoltre si scelgono le relazioni sintattiche tra le parole (forme attive o passive). In questo step si scelgono le nuance di significato che deve avere un termine (**lexical gap** per diverse lingue) (Es. partire va bene sia per treni che per aerei, ma decollare va bene solo per aerei)
- Referring expression generation: Si decidono le parole da usare per esprimere un'entità, l'obiettivo è bilanciare fluenza e ambiguità. Bisogna essere meno ambigui possibili, al fine di rendere i contenuti comprensibili e al contempo generare una frase fluida, che si possa leggere in modo naturale (Es. Mario o lo studente che sta svolgendo l'esame nel momento dell'orale coincidono e si possono usare in modo alterno per evitare ripetizioni e appesantire la frase)

Linguistic planning Faccio syntactic and morphological realization e orthographic realization, ampio la frase con conoscenze linguistiche per renderla coerente. Uso array al fine di generare la frase

- Syntactic and morphological realization: si applicano regole morfologiche e di sintassi (Es. aggiunta suffisso -ed per il passato in inglese e riordinamento di parole)
- Orthographic realization: si gestisce punteggiaturra, formattazione e capitalizzazione delle parole

9 Anatomia parser

I parser sono caratterizzati da 3 componenti:

- La grammatica che accettano
- L'algoritmo utilizzato (performance) (con la strategia di ricerca e l'organizzazione della memoria)
- L'oracolo utilizzato (probabilistico o basato su regole)

Se la strategia di ricerca è top-down si usa l'approccio **Razionalista**, in caso contrario si parla di approccio **Empiristico**. Nel corso abbiamo visto un solo parser Razionalista, che utilizza backtracking e un oraolo a regole, l'ambiguità strutturale del parser è espressa dalla serie potenza (numeri catalani), ma l'esplosione combinatoria del numero di alberi può essere gestita tramite programmazione dimanica. Il problema dell'approccio top-down è che genera moltissimi alberi "inutili" ai fini dell'analisi di una singola frase, inoltre la ricorsione a sinistra (tipica del linguaggio naturale) crea facilmente loop o overflow durante la computazione